Los Alamos National Laboratory

LA-UR-04-1652

Approved for Public Release
Distribution is Unlimited

Title: **Lightning: A Performance and Scalability Report on the use of 1020 nodes.**

Authors

Kei Davis
Adolfy Hoisie
Greg Johnson
Darren J. Kerbyson
Mike Lang
Scott Pakin
Fabrizio Petrini

Published **For distribution at LANL, and within the ASCI program**

CCS-3
Modeling, Algorithms, and Informatics Group

PAL

Performance and Architecture Lab
http://www.c3.lanl.gov/par_arch

Lightning: A Performance and Scalability Report on the Use of 1020 Nodes

Kei Davis, Adolfy Hoisie, Greg Johnson, Darren J. Kerbyson, Mike Lang, Scott Pakin, Fabrizio Petrini.

Performance and Architectures Laboratory (PAL),
Computer and Computational Sciences Division (CCS),
Los Alamos National Laboratory

1. Summary

This document contains a performance and scalability report based on the suite of measurements that were obtained on 1020 nodes of Lightning in late December 2003. Over a 2 day period, the PAL team was given access to 4 partitions of the Lightning system. Each partition contained 255 dual Opteron Nodes running at 2GHz.

The report is structured as follows:

- Single Node Performance, in particular memory performance in Section 2.
- System activity analysis in Section3, including the background "computational noise" that may impact application performance.
- Network Performance for a multitude of user-level communication tests in Section 4.
- Application Performance in Section 5 for Sweep3D (Sn transport), SAGE (hydro), and Partisn (Sn transport)

2. Single Node Performance

2.1. Memory Latency

To assess the memory performance within a single Lightning node we utilized a microbenchmark that measures the latency to memory from each processor. The microbenchmark spawns two threads. The first thread accesses one word per 64 bytes (the cache line size) of a memory region, measuring the average latency. Meanwhile, the second thread spins idly. When the first thread has finished measuring memory access latency, the two threads switch roles: the first thread spins idly while the second thread measures memory access latency. These tests are repeated for memory regions of sizes 4 KB to 1 GB to gauge the latency of each level of the memory hierarchy. The idea behind using multiple threads is to determine the difference in latency between memory accesses from the allocating CPU versus from the non-allocating CPU. Spinning helps inhibit thread migration, as there is no idle CPU to which to migrate a thread.

Figure 2.1 shows the latency measured for each level of the memory hierarchy. The microbenchmark reports that each CPU observes a 3-cycle latency (at a clock rate of 2 GHz) to access its 64KB on-chip L1 cache and a 12-cycle latency to access its 1MB on-chip L2 cache. Main-memory performance is trickier to report because the Lightning nodes are not symmetric multiprocessors (SMPs). Rather, each of the two CPUs is locally connected to half of the main memory and must remotely access the other half. Local memory sees an average of 134-cycle latency while remote memory sees a 206-cycle average latency—a penalty of over 50%.

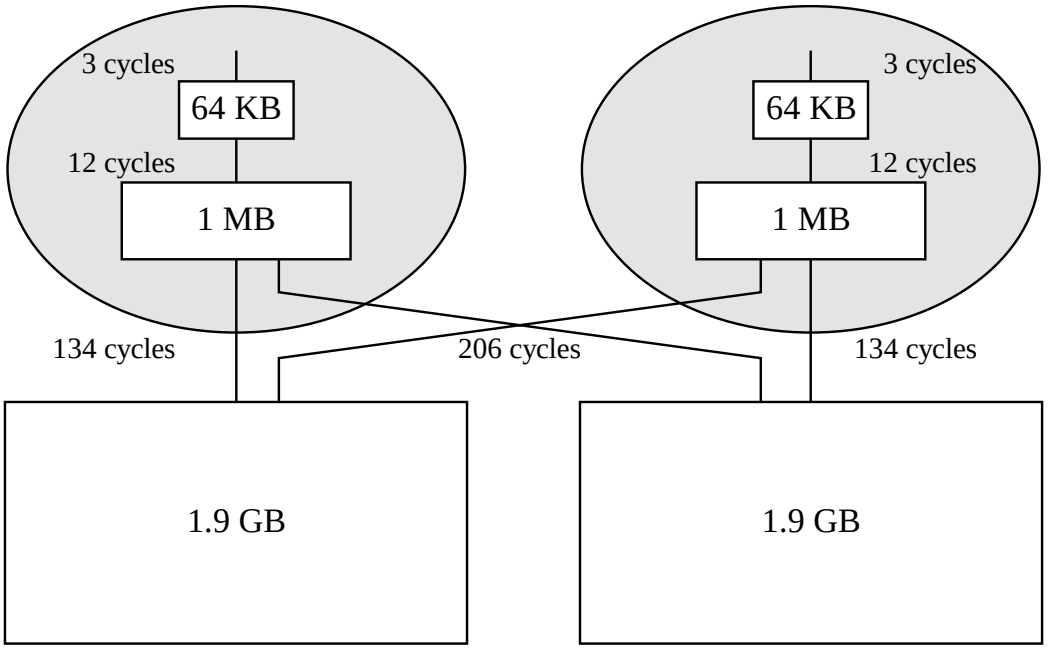

Figure 2.1 – Latency to the various levels of the memory hierarchy.

LANL's key applications use MPI and therefore follow the shared-nothing model. Ideally, therefore, all of the memory needed by a process can be kept locally, yielding the faster 134-cycle access time. However, the Linux kernel running on Lightning was not configured to take locality into consideration when allocating memory. As a result, the operating system arbitrarily scatters a process's pages across local and remote memory, leading to a mean memory latency somewhere between 134 cycles and 206 cycles.

Although a latency of 103ns (206 cycles @ 2 GHz) is still quite good in absolute terms, our analysis indicates that the latency can be reduced significantly, by more than 10%, by recompiling the kernel specifically for the Opteron and enabling NUMA support to keep allocations local.

2.2. Memory bandwidth

The bandwidth to memory was measured using cachebench. This benchmark performs one of 8 operations: reading, writing, read-modify-write (and tuned or manually unrolled versions of the same), memset and memcpy. These are performed for a certain sized data set thus stressing different parts of the memory hierarchy as the data set size increases. The results from executing cachebench on a single processor within a Lightning node are shown in Figure 2.2.

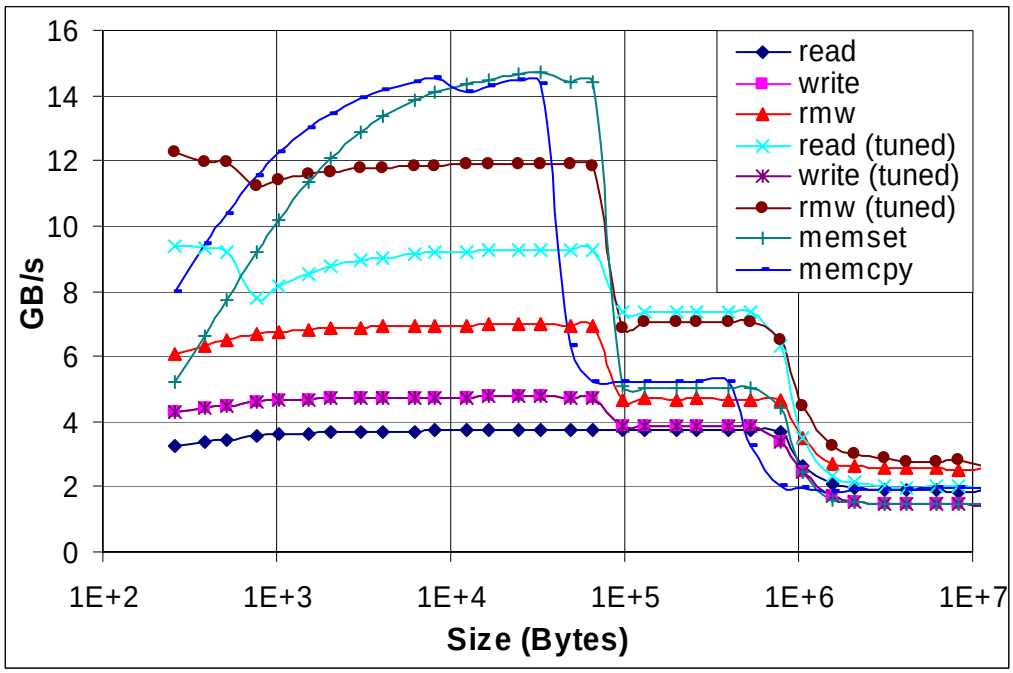

Figure 2.2 – Bandwidth to memory as measured by Cachebench

It should be noted that the bandwidth to main memory when reading is approximately 2GB/s and for writing is approximately 1.8GB/s. These are comparable with the ES45's in the Q system.

Cachebench was also run in the presence of a second task on the second processor which constantly read from main memory. Thus, this measured the contention within a node when both processors access memory. The relative performance between a single processor accessing memory (as shown in Figure 2.2) and when both processors are active to memory is shown in Figure 2.3. The Y-axis is the relative performance – a value of 1 means that there is no difference in performance between 1 processor and both processors accessing memory whereas a value less than 1 indicates contention. It can be seen that there is no impact when the data set size is less than the L2 cache size (as expected), and that the impact when both processors are accessing main memory is 6% when reading and 18% when writing.

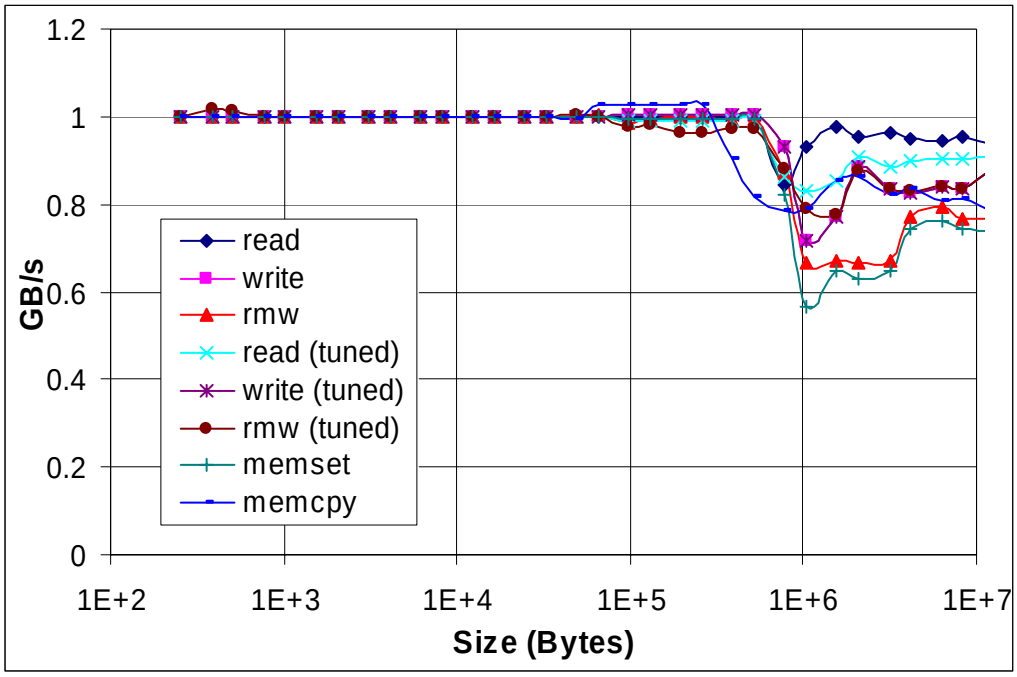

Figure 2.3 – Relative memory performance between 1 and 2 processors accessing memory within a node.

3. Computational Noise

Figure 3.1 shows the total slowdown, computed on a per-node basis, on the four clusters of Lightning. Overall the slowdown is contained, only 0.7%, and deterministic. Also, there is no asymmetry between the nodes.

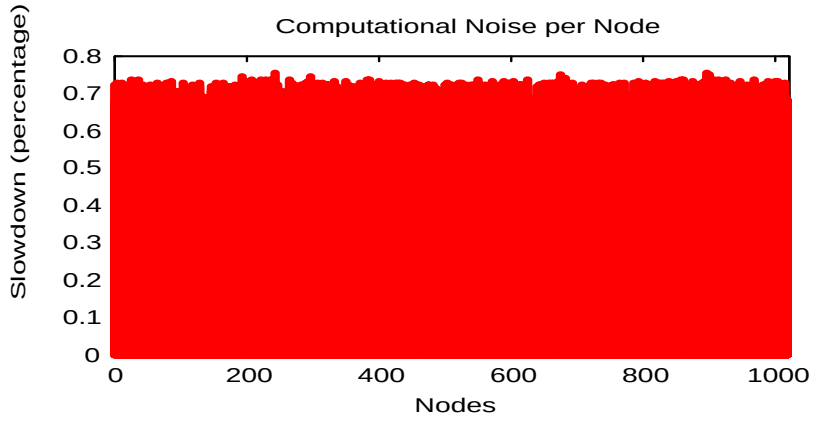

Figure 3.1. – Slowdown on each node due to Computation Noise.

In Figure 3.2 we analyze the noise on a node of Lightning. A first, preliminary analysis would suggest that there are at least five distinct sources of noise (harmonics) that could lead to a performance degradation of at most 10% on a large system, for applications that exhibit bulk-synchronous behavior and have a computational granularity of 1ms.

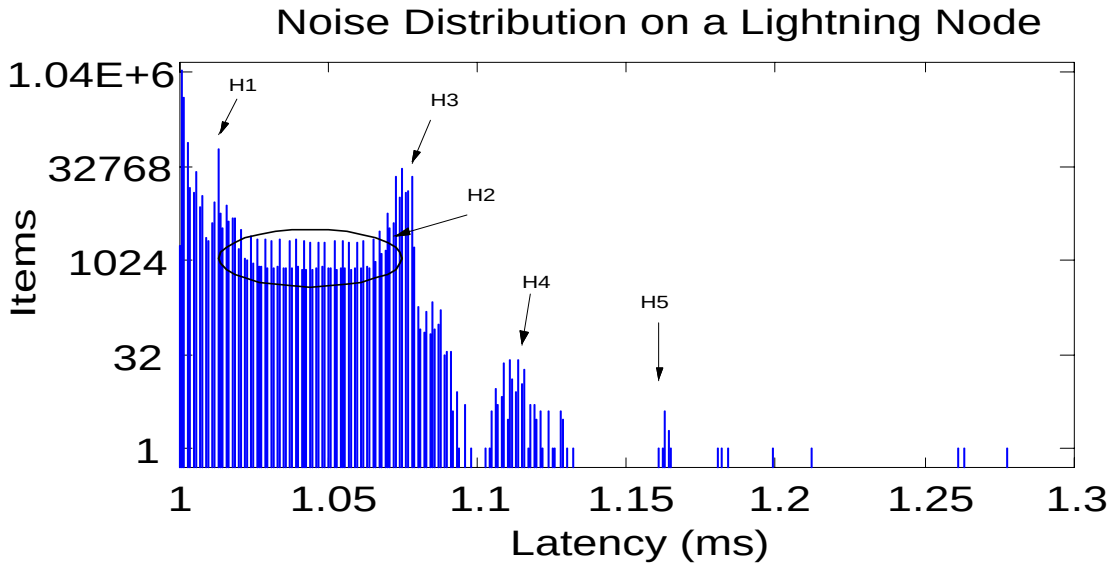

Figure 3.2 – Noise distribution on a single Lightning node.

4. Communication Performance

In this section we describe the performance of the MPI-CH version of MPI installed on Lightning at the time of our tests.

4.1. Unidirectional Performance

The network delivers a basic latency of 6.7 μs and an asymptotic bandwidth of 242 MB/s. These values are very close to the optimal performance that can be delivered by Myrinet. The data is plotted in Figures 4.1.1 and 4.1.2.

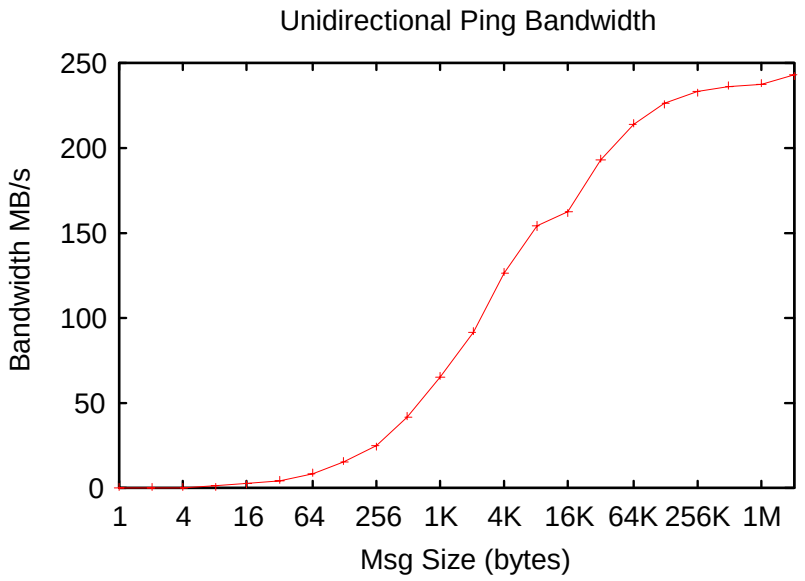

Figure 4.1.1 – Unidirectional Ping Bandwidth between adjacent nodes.

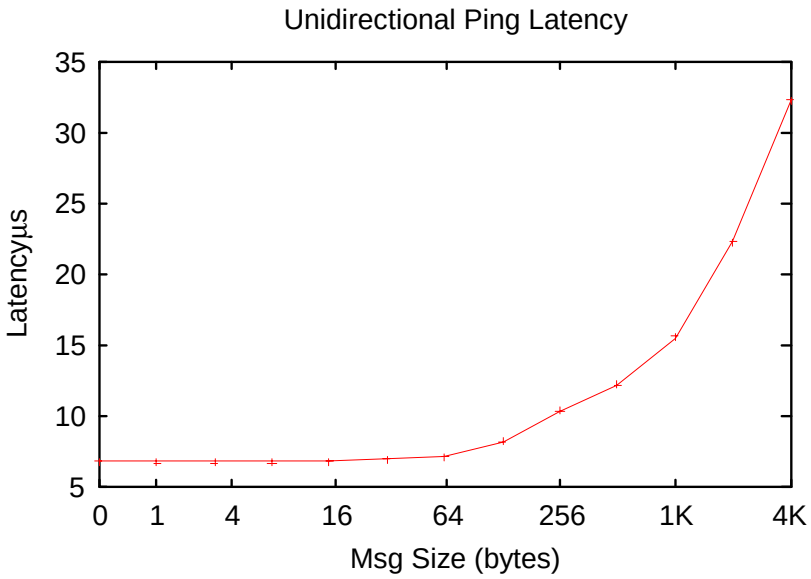

Figure 4.1.2 – Unidirectional Ping Latency between adjacent nodes.

4.2. Bidirectional Performance

The network is able to sustain bidirectional traffic without incurring any measurable performance degradation. The asymptotic bandwidth is about 240 MB/s and the basic latency 10 μs, as shown in Figures 4.2.1 and 4.2.2.

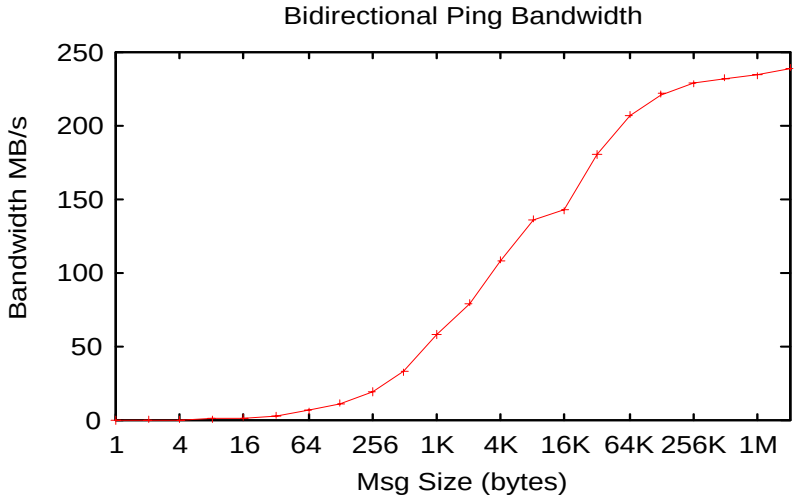

Figure 4.2.1 – Bidirectional Ping Bandwidth between adjacent nodes.

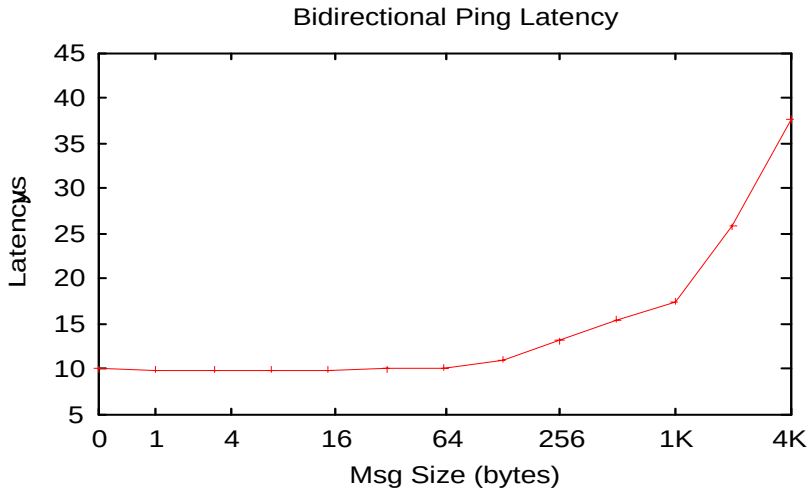

Figure 4.2.2 – Bidirectional Ping Latency between adjacent nodes.

4.3. Intra-node

The MPI-CH implementation obtains excellent in-box performance with 900 MB/s of bandwidth and only 1µs of latency (Figure 4.3.1 and 4.3.2).

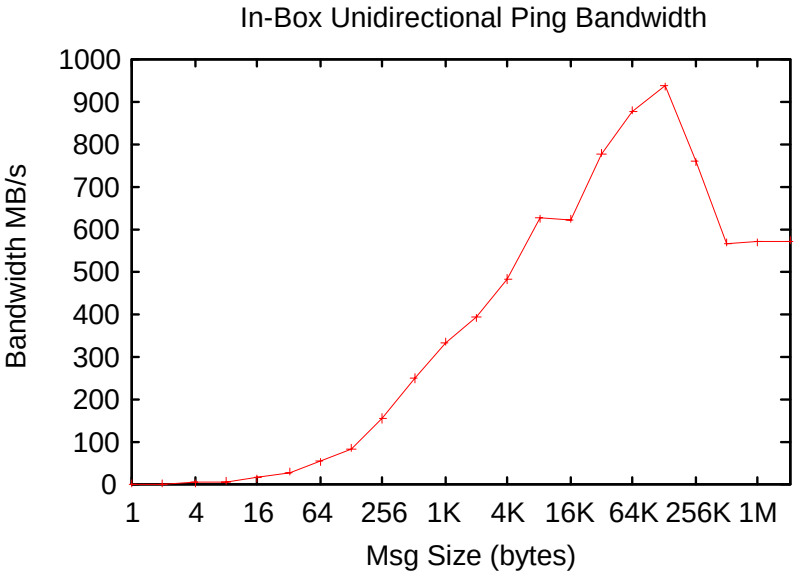

Figure 4.3.1 – In-box Unidirectional Ping Bandwidth.

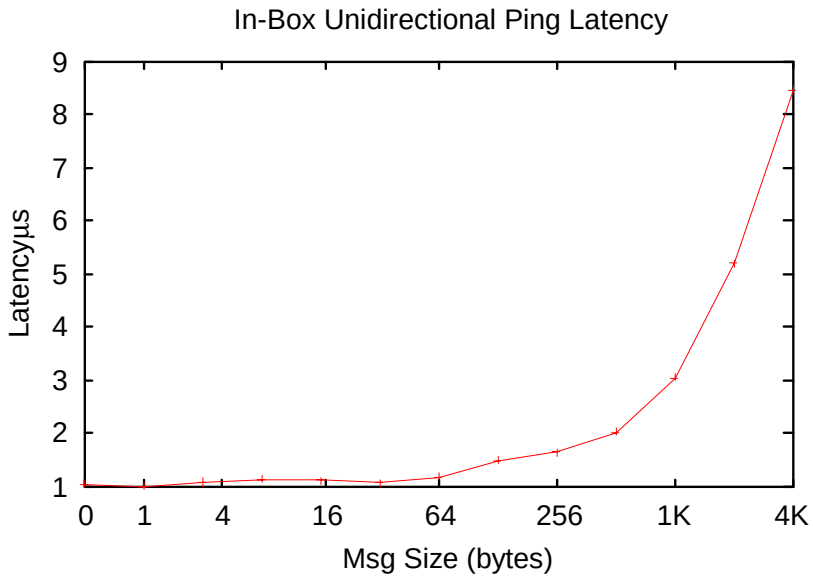

Figure 4.3.2 – In-box Unidirectional Ping Latency.

4.4. Distance

As shown in Figure 4.4, the network is almost insensitive to node distance. The graph doesn't report all nodes, because we couldn't collect all data due to the benchmark crashing multiple times.

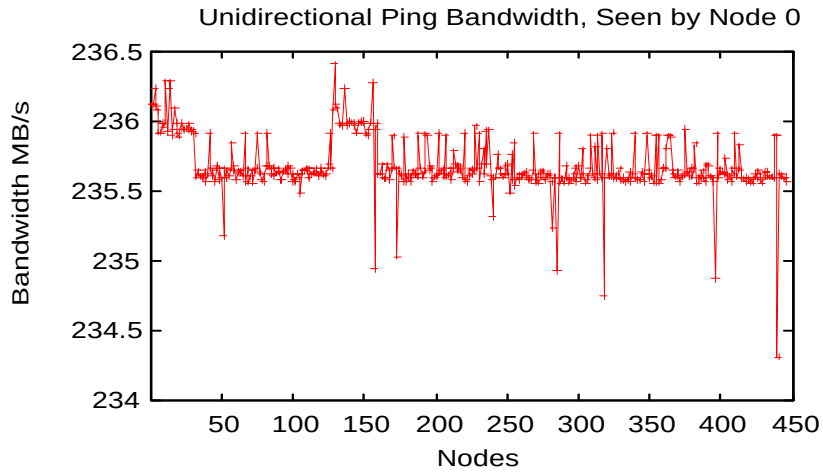

Figure 4.4 – Unidirectional Ping Bandwidth from Node 0 to all other nodes.

4.5. Hotspot

As seen in Figure 4.5, the network delivers acceptable performance under hot-spot traffic, up to 64 nodes. With more communication partners the performance degrades gracefully down to 110 MB/s.

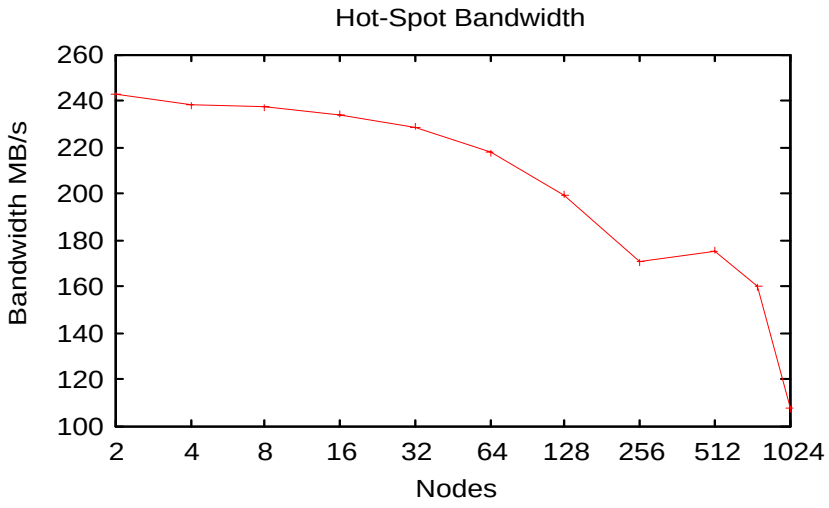

Figure 4.5 – Hotspot bandwidth.

4.6. Complement Traffic

In the complement permutation pattern each node sends messages to a partner node which is identified by the complement of its bit string. For example, on a 512-node machine, node 0 sends messages to node 511, node 1 to node 510, etc. The complement traffic exposes the aggregate network performance as a whole, because it stresses many network routes, the routing algorithm and the flow control. As seen in Figure 4.6, only three pairs reach the full bandwidth. In all other cases we get sub-optimal values, with an average of 100 MB/sec. This is a feature of the Myrinet's routing algorithm, which is deterministic. In fact, for each pair of communicating processors, a static route is chosen at initialization time by a mapper, and this leads inevitably to congestion in the network. This experiment emphasizes the need for adaptive routing, where the network switches are able to identify local congestion and use alternative routes.

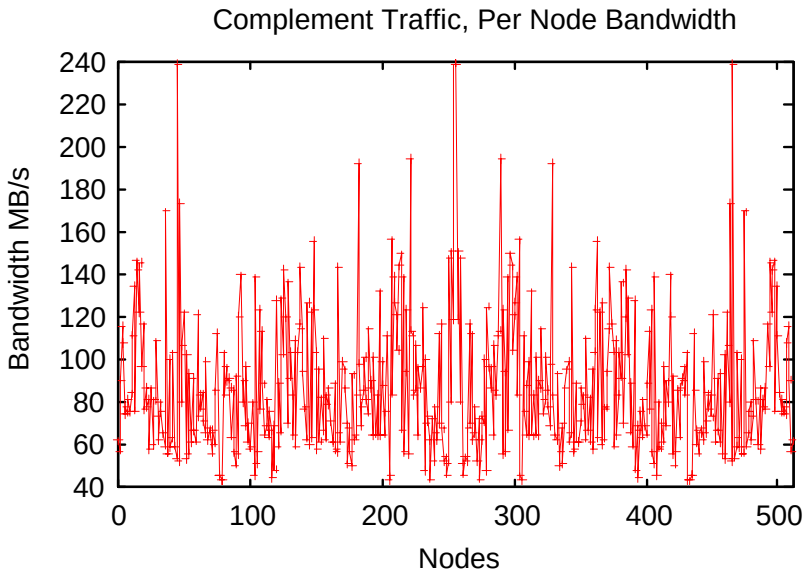

Figure 4.6 – Complement bandwidth.

4.7. Collectives

Both barrier and allreduce, two important collective patterns for the ASCI workload, scale logarithmically. The latency on the whole machine is about 250 µs, when we use both processors in each node. The gap between the two and one processor graphs, almost a factor of two at scale, is due, in part, to the noise in the system.

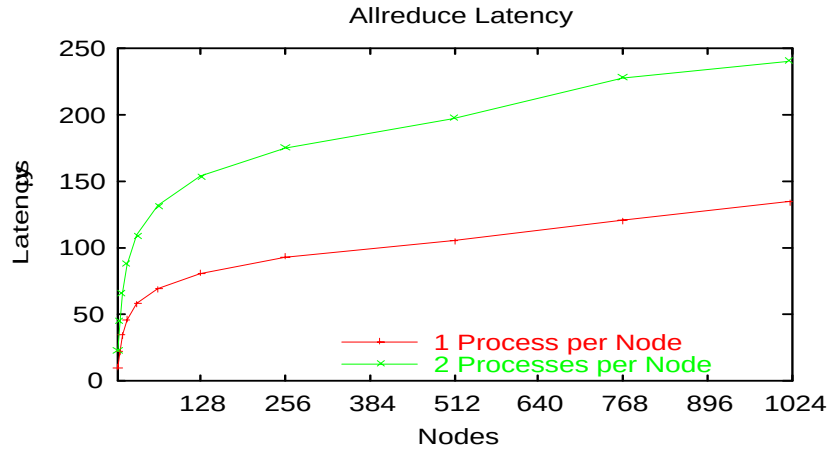

Figure 4.7.1 – Allreduce Latency.

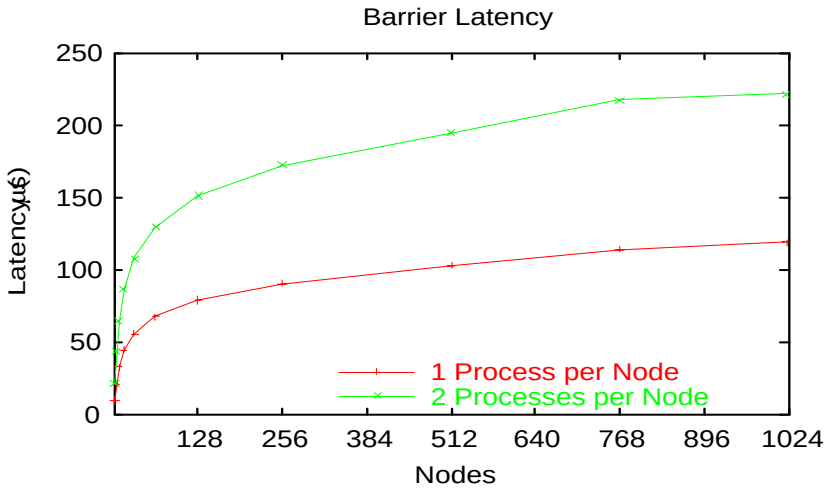

Figure 4.7.2 – Barrier Latency.

4.8 Elimination of Routing and Network Problems

In the initial phase of our tests we were able to pinpoint a serious performance problem that was reducing the communication performance by a factor of four, in the worst case. A small number of nodes connected to a common crossbars were mis-routed to the upper-level switches (figure 4.8). This resulted in congestion in what should have been contention-free communication. The test was simple but representative of communication patterns in the ASCI workload. It was later determined that the problem was related to incorrect cabling and has since been corrected. Initial investigation with SAGE showed a improvement of up to 10% on 100 processors.

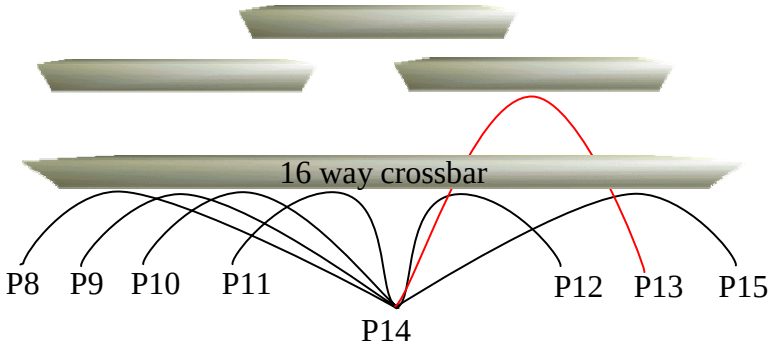

Figure 4.8 Erroneous routes to nodes on common cross-bar

5. Application Performance

5.1. Sweep3D

The performance of Sweep3D was measured on Lightning on two problem sizes, and for each size using 2 blocking factors (four input decks total). Problem sizes of 50x50x50 cells and 5x5x400 cells per processor were used in a weak-scaling mode. The blocking was chosen either using 1-plane in the Z-direction (k-plane)/1 angle, or 10 k-planes/3 angles per block. The performance of these four cases was measured when using one processor per node (ranging from 1 to 1,020 processors) and when using both processors per node (ranging from 2 to 2,040 processors). The intention of these experiments was to determine how well Sweep3D performs and how well it scales for both coarse- and fine-grained problems, smaller and larger messages, and with and without memory/network contention.

The performance shown by the curves in Figures 5.1.1 to 5.1.4 are all quite reasonable for a system with Lightning's architecture. Each figure contains two curves showing the performance of Sweep3D when using either 1 processor or 2 processors per node.

Figure 5.1.1, the 50x50x50 case with 1 k-plane/1 angle blocking, shows a fairly flat performance with only slight overhead penalties at larger processor counts. The difference when using both PEs per node to the case when using only 1 PE can be seen to be approximately 10%, i.e. the overhead when using both PEs results in a 10% increase in runtime but a doubling of the number of processors used.

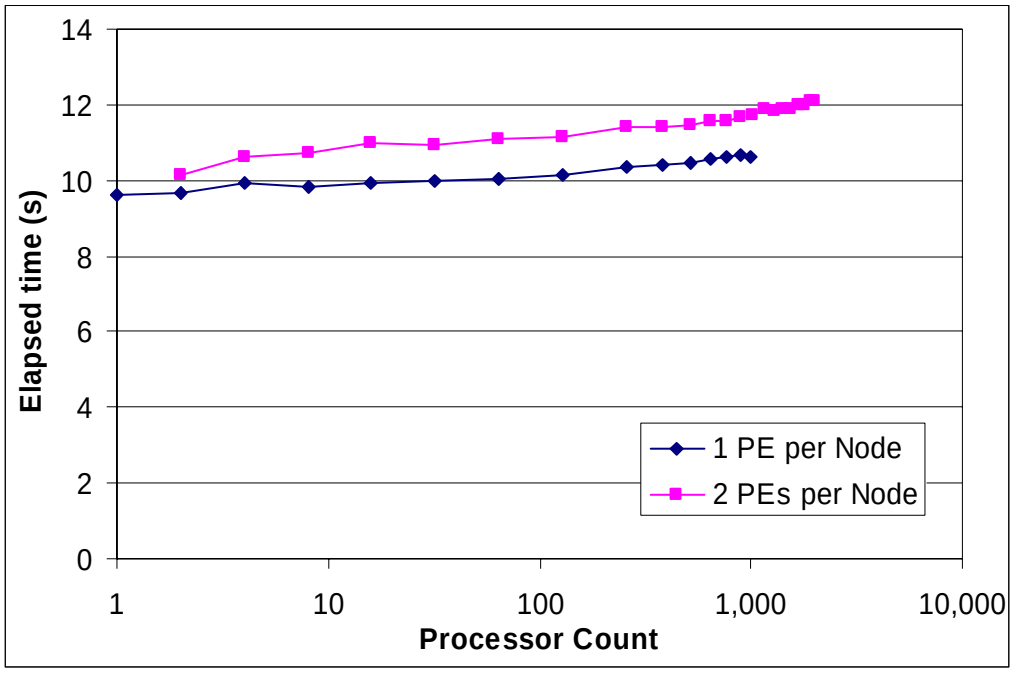

Figure 5.1.1 – Sweep3D performance on Lightning (50x50x50, MK=1, MMI=1)

Figure 5.1.2, the 50x50x50 case with 10 k-planes/3 angles blocking represents the most coarse-grained test. The poor scaling is, in fact, caused mainly by the complex behavior of Sweep3D itself and is not an idiosyncrasy of Lightning. There exist many idle processors during the sweep processing in this test, a result of the small number of blocks in the k-dimension, in turn caused by the large blocking parameters.
For the subgrid size of 50x50x50 cells per processor, a higher performance is achieved with 1 k-planes/1 angle per block than with 10 k-planes/3 angles per block.

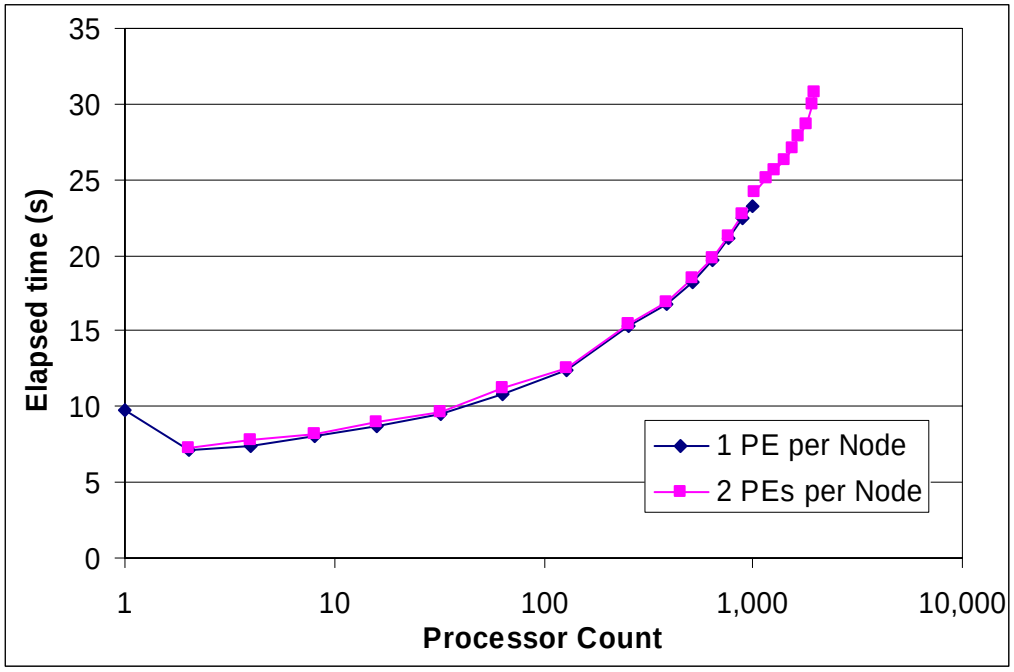

Figure 5.1.2 – Sweep3D performance on Lightning (50x50x50, MK=10, MMI=3)

Figure 5.1.3, the 5x5x400 case with 1 k-plane/1 angle blocking, shows a large performance penalty as the machine size scales upwards. While a single-processor run completes in 1.6 seconds, a 1020-processor run takes 6.6 seconds or a factor of 4.1 times longer to complete. At 32 processors and above, there is a constant gap (~2.2 seconds) between the 1 PE per Node curve and the 2 PEs per Node curve. This is likely caused by contention for the network. Starting at 32 CPUs, there are CPUs that both send and receive two messages on each sweep. The likelihood that the two CPUs in some node both try to send simultaneously therefore increases significantly at 32 CPUs. Because Sweep3D is tightly coupled, any contention slows down the entire run.

Figure 5.1.4, the 5x5x400 case with 10 k-plane/3 angle blocking, behaves largely as expected for Sweep3D running in a weak-scaling mode: runtime gradually increases with the processor count. Note that the blocking of the 5x5x400 case with 10 k-planes/3 angles per block results in a much better scaling performance than the same case with the 1 k-plane/1 angle blocking shown in Figure 5.1.3. The former blocking parameters result in a reduced number of messages between processors with each message having a higher

payload than for the latter case. The latter case thus has a higher number of messages with a smaller payload and is thus more sensitive to the network latency.

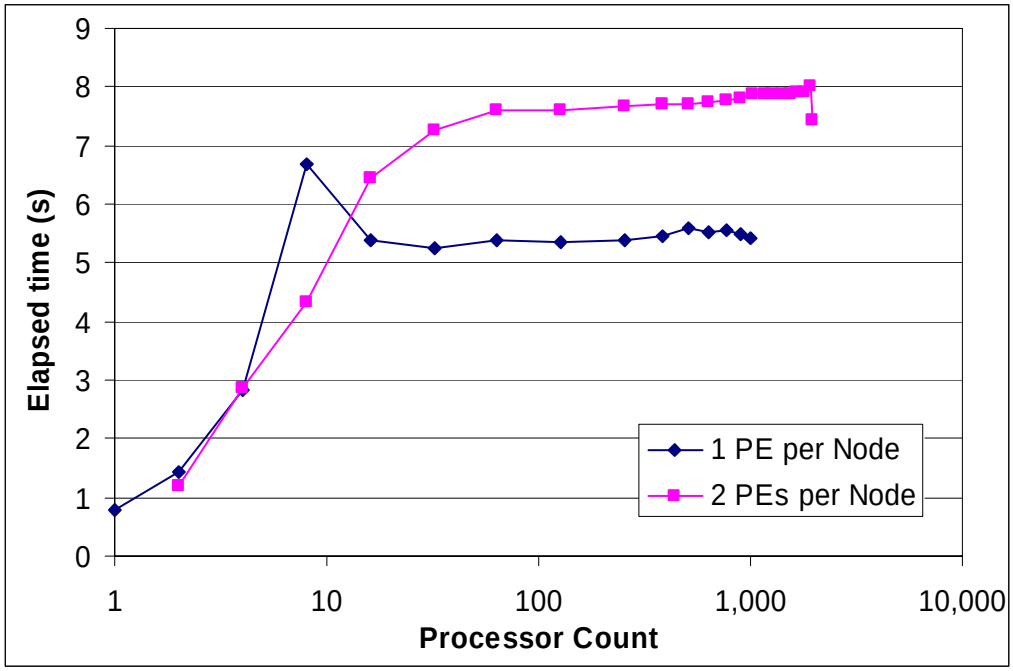

Figure 5.1.3 – Sweep3D performance on Lightning (5x5x400, MK=1, MMI=1)

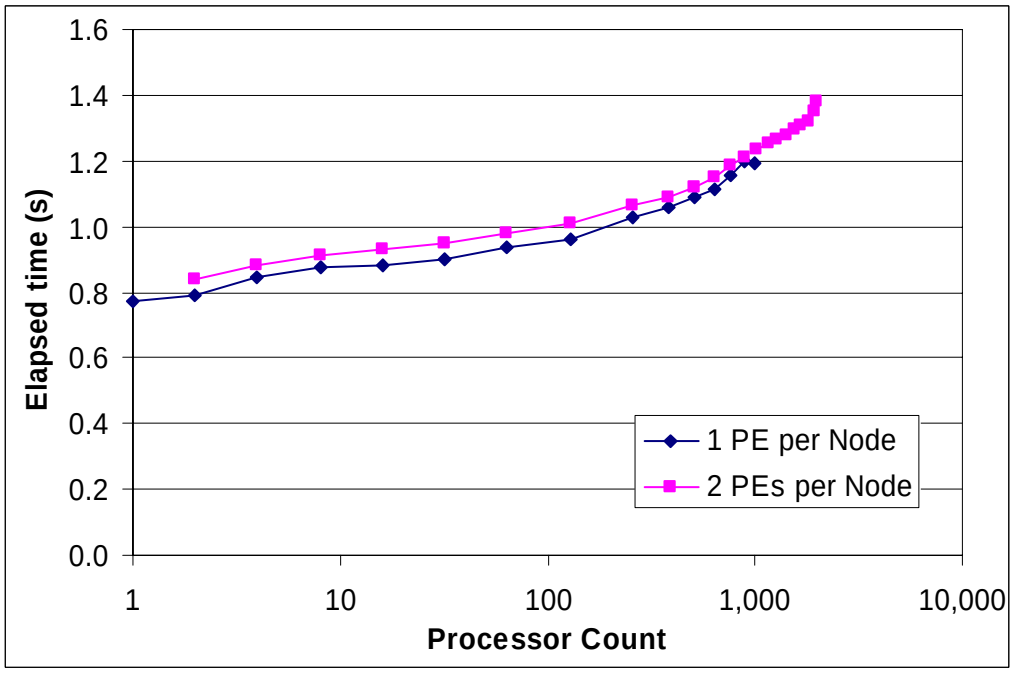

Figure 5.1.4 – Sweep3D performance on Lightning (5x5x400, MK=10, MMI=3)

5.2. Sage

Several tests of SAGE were performed on the available 1020 nodes of Lightning. The initial test examined the single processor and single node performance of the Lightning system. This was undertaken using both SAGE versions v20030505 and v20001220. Four input decks were used in this study: timing.input, timing_a.input, timing_b.input, timing_c.input, and timing_h.input. SAGE was compiled using both the PGI compiler and the Absoft compiler. In general it was found that the PGI compiler resulted in a performance on SAGE which was 10% higher than when using the Absoft compiler.

To examine the scaling behavior of SAGE, two tests were done. The first test used SAGE v20001220 with the timing.input deck which was done to stress the communication aspects of the application. This input deck has 13,500 cells per processor and does not use the solver. SAGE v20001220 was used in this test to allow a comparison with the SAGE performance already reported for ASCI Q. The second test used SAGE v20030505 with the timing_h.input deck which has the solver turned on and processes 35,000 cells per processor.

Figure 5.2.1 shows the single processor and single node performance of SAGE v20030505 on four input decks. The performance metric used on the Y-axis is the number of cell-updates per second per processor (cc/s/pe). It can be seen that when using both processors per node, the performance per processor degrades by 15% to 20%.

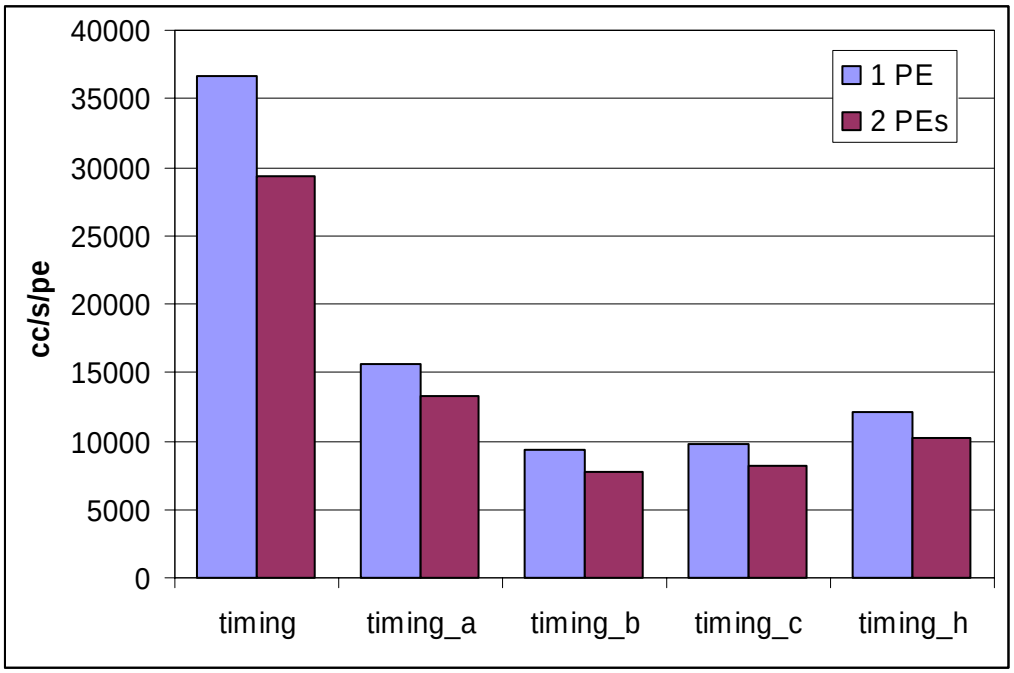

Figure 5.2.1 – Lightning single processor and single node performance of SAGE.

Figure 5.2.2 shows the scaling performance of SAGE v20001220 using the timing.input deck on Lightning when using either 1 processor per node or both processors per node. Here the metric used on the Y-axis is the time taken to perform a single iteration of SAGE (cycle time in seconds). Overall the sage scaling behavior is good – when using

both processors per node, the cycle time increases from 0.41s to 0.77s. This represents an efficiency at 2040 processors of 53%.

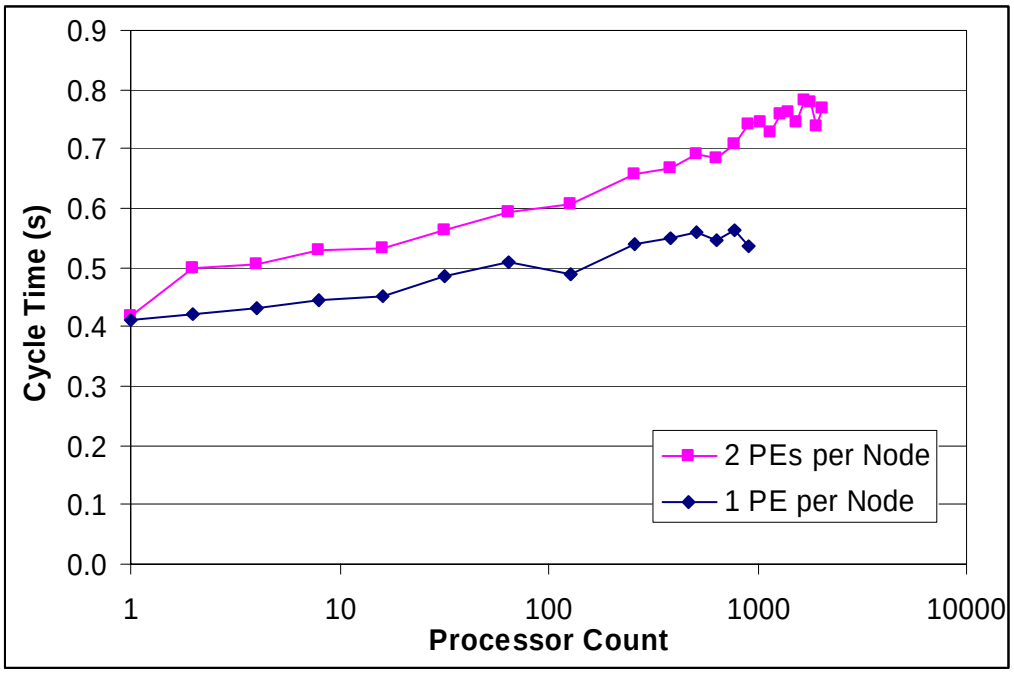

Figure 5.2.2 – SAGE v20001220 performance on Lightning (timing.input).

Figure 5.2.3 shows the scaling performance of SAGE v20030505 using the timing_h.input deck on Lightning when using either 1 processor per node or both processors per node. On this input deck it can be seen that the cycle time increases from 3.0s to 9.0 at 2040 processors representing an efficiency of 33%.

Figure 5.2.4 shows the relative performance of using both processors per node in comparison to using only 1. Note that the x-axis is the node count. For both the two scaling sets of data included in Figure 5.2.2 and 5.2.3, the relative performance is between 1.7 (when using 1 node) and 1.3 when using all 1020 nodes. The difference when using only 1 node is due to contention within the node. The difference when using all 1020 nodes is due to both contention within the node, and contention when sharing the communication network from each node. The maximum number using this relative performance metric is 2, i.e. using both processors is at best twice as good as using only one processor per node.

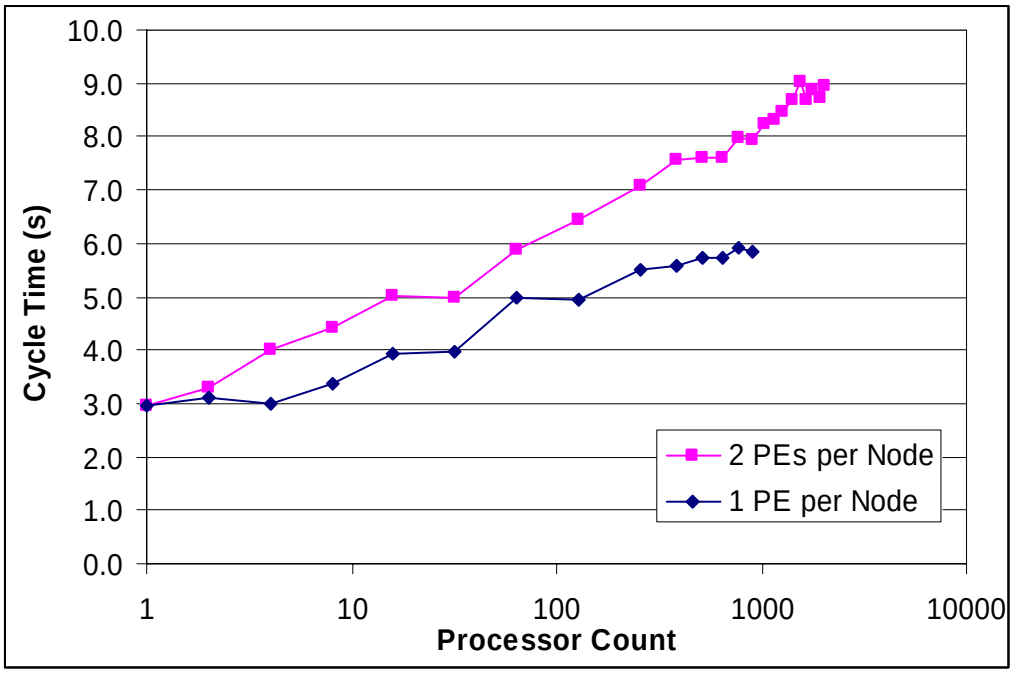

Figure 5.2.3 – SAGE v20030505 performance on Lightning (timing_h.input).

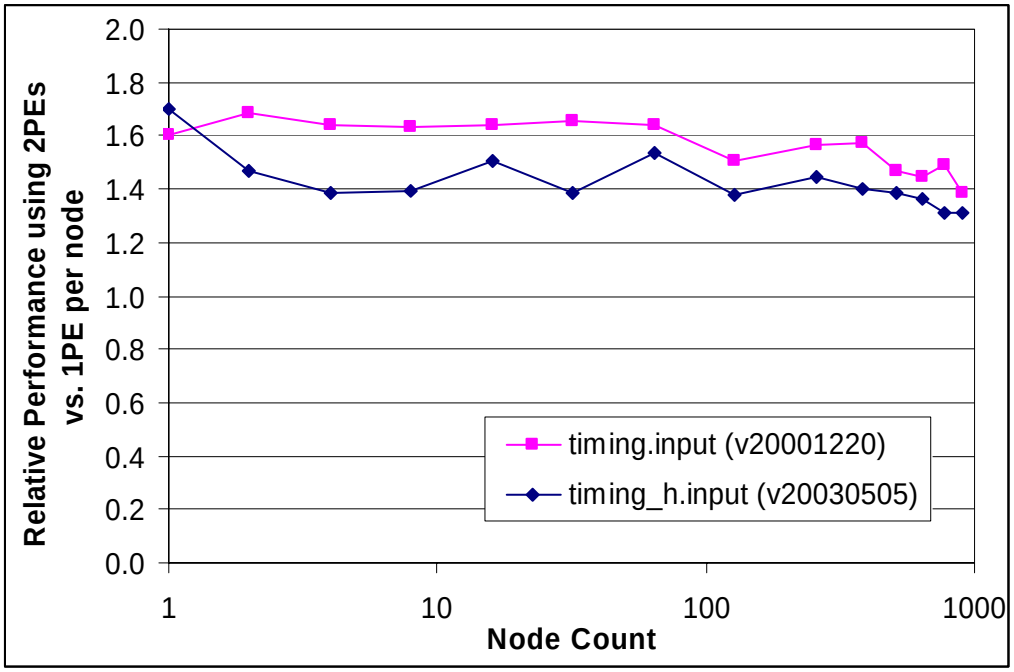

Figure 5.2.4 – Effectiveness of using 2 processors per node on SAGE.

5.3. Partisn

The performance of Partisn (version 3.02) was examined when using four input decks. The first input deck, sntiming, is the default testing input deck as included in the Partisn distribution. This input uses a 3-D spatial cube which grows in size with the processor count and maps approximately 13,000 cells to each processor. A modified version of this input, sntiming5x5, maps a subgrid of 5x5x400 cells to each processor irrespective of the processor count. Two further input decks, Rep1 and Rep15x5 correspond to sntiming and sntiming5x5, but have an increased processing requirement – having a greater S_N order (S_8 in comparison to S_6).

The total runtime for each of these input decks was obtained and is plotted in Figure 5.3. It can be seen in Figure 5.3 that the Rep1 and Rep15x5 have an increased processing time in comparison to their corresponding sntiming inputs. Since all input decks are for a constant problem size per processor (weak scaling), the ideal runtime is a constant. However, the runtime does increase as a result of scaling. The runtime at 2040 processors is between a factor of 3.2 (Rep15x5) and 11.3 (sntiming) longer than on a single processor. All results are reported when using both processors per node. The apparent jagged look to the 5x5x400 sub-grid cases are not unusual being due to the processing requirements of the solver.

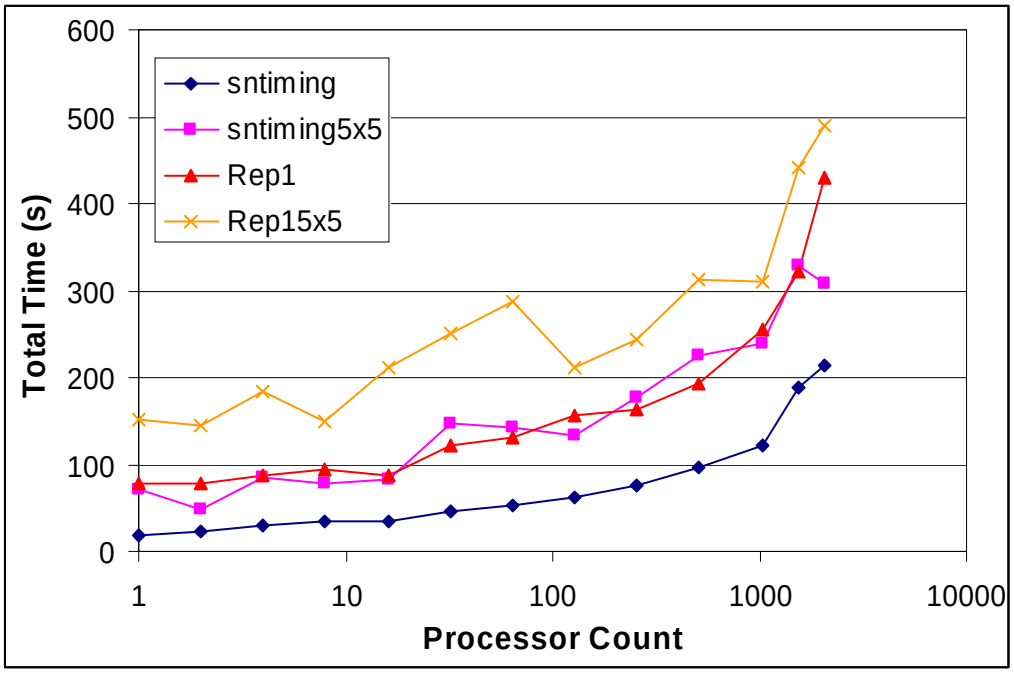

Figure 5.3 – Partisn performance on Lightning on four input decks.

6. Summary

We have benchmarked Lightning in its incarnation as of mid-December 2003, up to the full configuration of 1020 nodes.

The overall performance and scalability of the machine is in line with the expectations given the actual hardware configuration of the cluster, and given that the hardware and software are still being deployed and refined.

We summarize below some of the performance problems we identified, in particular those deficiencies that could be solved in order to improve the overall performance of the machine.

The Linux kernel running on Lightning was not configured to take locality into consideration when allocating memory. As a result, the operating system arbitrarily scatters a process's pages across local and remote memory within a node, as described in Section 2.1. Our analysis indicates that the latency can be reduced significantly, by more than 10%, by recompiling the kernel specifically for the Opteron and enabling NUMA support to keep allocations local.

We have found serious problems with the network, as described in Section 4.8. Debugging of the network took a significant amount of resources, after the actual running of the tests. This required devising microbenchmarks, analyzing large amounts of data and proposing solutions and helping with their implementation by working together with CCN-7 and Linux Networks. As of the release of this report most of the hardware problems with the network have been eliminated.

The performance of the collectives, described in 4.7, is negatively impacted by the network design and by the existing "computational noise" (Section 3) in the system. Further investigation on the noise is under way in collaboration with CCS-1.

All of the relevant performance data (network/communication and apps) was collected using MPI-CH available at the time of the data collection. No LA-MPI numbers are included.

The performance of the applications tested is in-keeping with the current status of the system.